Collided Path Replanning in Dynamic Environments Using RRT and Cell Decomposition Algorithms

Ahmad Abbadi(✉) and Vaclav Prenosil

Department of Information Technologies, Faculty of Informatics, Masaryk University, Botanicka 554/68a, 602 00 Brno, Czech Republic
Ahmad.Abbadi@mail.com, prenosil@fi.muni.cz

Abstract. The motion planning is an important part of robots' models. It is responsible for robot's movements. In this work, the cell decomposition algorithm is used to find a spatial path on preliminary static workspaces, and then, the rapidly exploring random tree algorithm (RRT) is used to validate this path on the actual workspace. Two methods have been proposed to enhance the omnidirectional robot's navigation on partially changed workspace. First, the planner creates a RRT tree and biases its growth toward the path's points in ordered form. The planner reduces the probability of choosing the next point when a collision is detected, which in turn increases the RRT's expansion on the free space. The second method uses a straight planner to connect path's points. If a collision is detected, the planner places RRTs on both sides of the collided segment. The proposed methods are compared with the others approaches, and the simulation shows better results in term of efficiency and completeness.

Keywords: Path re-planning · Motion planning · RRT · Cell decomposition · Multi RRT

1 Introduction

The motion-planning problem is an active subject in the robotics field. It attracts the researchers to develop and increase the motion independence of the systems. The high demand of autonomous system leads to develop many concepts of motion planning. They vary in the efficiency and the domain of applications. Examples of these algorithms are: local planners, e.g. Bug algorithm [1]; roadmap approaches, for example, visibility roadmap and Voronoi diagrams [2]; cell decomposition methods, which are divided the working space into manageable regions, and classified the workspace into free and obstacle areas. Other examples of motion planners are the randomized sample-based algorithms, these approaches try to approximate the workspace by taking samples from it randomly [3]. Recently, many researchers have studied the combination of these methods, in order to avoid the drawbacks and enhance the performance.

In this paper, our focus is to develop an efficient planner on slightly changed workspace. The proposed methods are designed for the robot's omnidirectional movement. The approximation cell decomposition (ACD) is used and combined with the RRT planner in order to enhance the robot's navigation. The ACD finds a spatial path on

J. Hodicky (Ed.): MESAS 2015, LNCS 9055, pp. 131–143, 2015.
DOI: 10.1007/978-3-319-22383-4_9

preliminary and stationary workspaces, and then the RRT is used to validate this path on the actual workspace.

Two methods have been proposed in this paper. The First one creates instances of RRTs and biases their growth toward the points of the ACD's path in ordered form. It updates the bias values based on the collision information. If changes are made to the workspace and a collision is found along the path's segments, the planner attempts to find a new sub-path locally by exploring the space around the collision place using RRT. The Second method uses a straight-line planner to connect the path's points. It creates local RRT trees on both sides of the collided segment of the path, if one is detected.

This paper organized as follows, The RRT algorithms and related developments are presented in Sect. 2. In Sect. 3, the principle of cell decomposition algorithms is reviewed. The proposed method and result is discussed in Sects. 4 and 5, respectively. Finally, we conclude the results.

2 Rapidly Exploring Random Tree (RRT)

The rapidly exploring random tree algorithm is a sample-based motion planner [4, 5]. It does not evaluate the workspace in an exact manner; rather, it deals with configurations that are taken randomly from the configuration space. The principle of RRT algorithms is to build a filling space tree and pulling the tree's growth toward unexplored regions. It takes a configuration randomly and then branching a new extension from the nearest node of the tree toward this configuration. The new branch's length is determined by the incremental step parameter. If this branch does not collide with obstacles or it does not break constrains, it is kept in the RRT tree. Thus, the tree is growing outward of the initial position. The principle of the basic RRT algorithm is shown in Fig. 1.

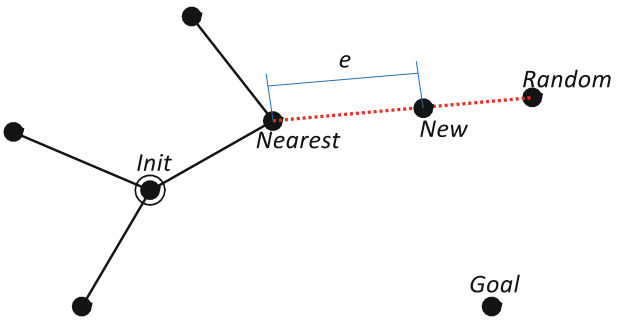


Fig. 1. RRT principle

In navigation problem, the RRT produces a feasible path between the initial position and the goal one, if these locations existed in the tree's nodes. The feasibility of the path comes due to the tree characteristic, where the tree is built up of valid connections between the tree's nodes. It starts from the initial location and explores the space until

it finds the goal, then it produces the path. The tree's nodes from the root to the goal leaf represent the path's vertices.

The basic RRT algorithm is shown in Fig. 2. It takes as input parameters the initial location, the goal location, and the incremental step. In addition, it takes termination parameters such as a maximum number of attempts to grow branches, a time limitation, or other parameters based on the application. The output is a graph has a tree structure, where the nodes represent the tree's vertices, and the edges represent the connection between these vertices.

```
Algorithm: RRT
Input:   Initial, Goal, Max Iteration I,
         incremental distance ε.
Output:  The tree graph G.
1.  G:init(Initial)
2.  FOR (i = 1 TO I) BEGIN
3.    randomPnt = randConfiguration()
4.    nearestPnt = G.nearestVertex(randomPnt)
5.    newPnt = NewConfiguration(nearestPnt,randomPnt,ε)
6.    IF NOT isCollided(nearestPnt,newPnt) BEGIN
7.       G.addVertex(newPnt)
8.       G.addEdge(nearestPnt,newPnt)
9.       IF G.checkGoal(Goal)BEGIN
10.        RETURN G.success()
11.      END
12.   END
13. END
14. RETURN G.fail()
```

Fig. 2. RRT algorithm

The algorithm starts by placing the tree's root on the initial location, and then it takes a random sample from the configuration space. It finds the nearest tree's vertex to this sample, and creates a new point on the segments between the chosen random point and the nearest point. The new point is located far from the nearest point by a distance equals to the incremental step. If no collision is detected, then the algorithm adds the new point as a vertex to the tree and the segment between the new point and the nearest vertex is added as an edge to the tree. These steps are repeated until a path between the initial and the goal locations is found, or a termination condition is satisfied.

RRT algorithm attracts the attentions due to its simplicity and its success in solving the complex navigation problems, including the problems that have dynamic and kinematic constraints. In the next paragraphs, some of RRT developments and improvements are reviewed.

The basic RRT planner grows one tree and tries to find the goal point. A development of this approach proposed the use of bi-directional trees or multi-trees. These trees bias toward each other in order to merge and form united structure. This strategy enhances

the possibility to find a route more quickly because instead of searching for one point, the goal one, any connection with the others trees' nodes can lead to a solution [6, 7].

The second category of RRT improvements based on the changing of sampling strategies; some studies introduce the bias toward the goal configuration, which means choosing the goal location by a specific value of probability instead of taking a random sample. Other researchers suggest making the bias toward hull around the goal [5], or to previous configurations of the success plans [8, 9]. A survey of RRT variations and developments were reviewed and published in [10].

The main drawbacks of RRT algorithm appear when it operates in small and narrow area, due to the random sampling strategy. The basic RRT uses a pseudorandom sample generator. This uniform distribution takes a sample from the space in equal probability, which means, the small regions have a lower probability to sample within them. As a result, the RRT efficiency is decreased when the workspace contains narrow areas.

3 Cell Decomposition

The key idea behind the Cell decomposition algorithms (CDs) is to divide the workspace into manageable regions. These regions are classified into two categories, the areas located in the obstacles space (The obstacle cells) and the areas in the free space (The free calls).

In navigation application, the CDs are utilized to find a path through the free cells. In order to simplify the navigation problem, these algorithms build a graph of the adjacent free cells to represent the free workspace. The graph's nodes represent the free cells, while the graph's edges represent the adjacency relation between the cells; two adjacent cells, which share a common barrier, create an edge in the graph.

The CDs approaches are classified as exact methods and approximation ones. In the exact cell decomposition cases, the free workspace is equal to the union of all generated cells exactly, while in approximation methods the free workspace is approximated by set of adjacent free cells.

An example of exact cell decomposition methods in 2D is the trapezoidal cell decomposition as shown in Fig. 3-a. It creates striped trapezoidal or triangular cells by means of sweeping line technique. Figure 4, shows the graph of adjacency to the example in Fig. 3-a, the shaded boxes represent the route of free cells between the initial and the goal locations.

The other examples of exact cell decomposition methods and its application are proposed and discussed in [1, 3, 11, 12].

The quad-tree approximation algorithm is an example of approximation cell decomposition (ACD) in 2D [2, 13]. It divides the workspace into four quarters. If a quarter locates in the free areas completely, it is marked as a free cell; otherwise, it is marked as an obstacle cell if it locates in obstacles areas completely. The other case when this quarter contains parts of both free and obstacles regions, in this case, the algorithm divides it into four quarters. This process is repeated until all cells are located completely either in free areas or obstacle regions, or a specific resolution is reached. The resolution in this case represents the smallest cell's edge. Figure 3-b shows the quad-tree cell

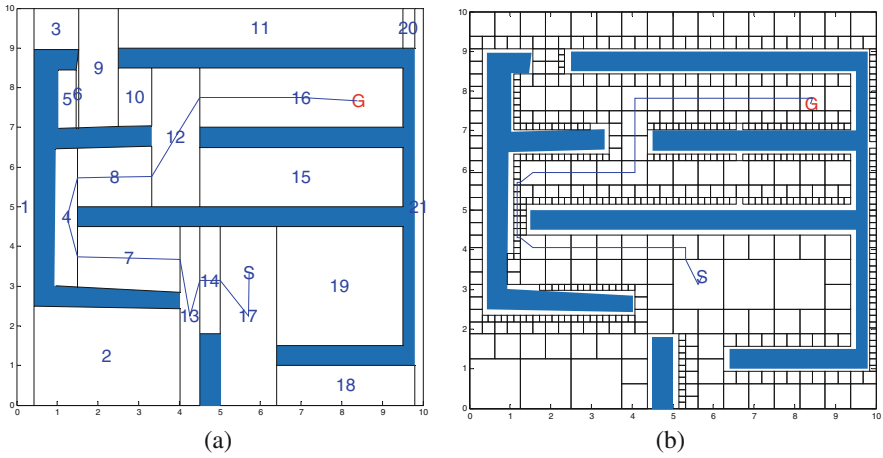


Fig. 3. The free space representation using, a: an exact CD method, b: an approximation CD method

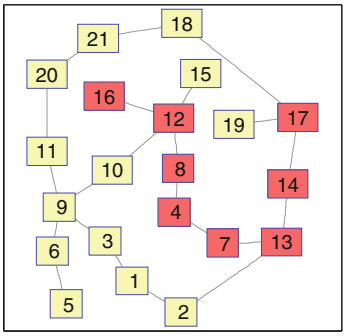


Fig. 4. Cell decomposition's adjacency graph. The dark boxes represent the free route in the workspace between the initial's cell and the goal's cell

decomposition methods. Another example of the approximation cell decomposition in navigation problem is discussed in [14–17].

The main drawback of CDs methods is the sensitivity to the environment changes. Where any small changes require re-executes the algorithm again and generates another adjacency graph.

4 Proposed Methods

In this work, RRT and ACD algorithms are combined together in order to exploit the advantages of each of them. The new planners try to overcome the drawbacks, which affect the performance of the navigation process significantly, by complementing these two approaches. The RRT planner has relatively high tolerance to obstacles shapes and workspace changes. This feature is missing in ACD planner. In addition, The RRT is

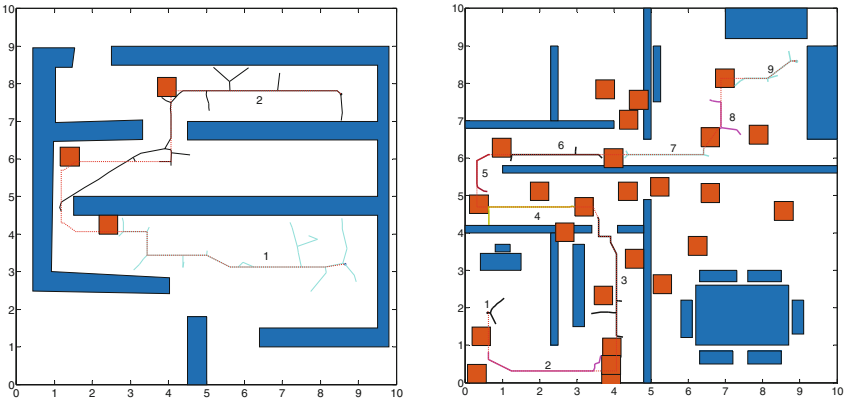


Fig. 5. The Proposed methods. The dotted line represents the ACD's path on stationary workspace. a: The RRT validator method creates two RRT trees from the initial and the goal locations. b: The local RRTs method creates nine local RRT trees

not effective in small areas or narrow passage, while ACD planner does not face this problem. Based on that, the efficient spatial planner, ACD, is used to plan a primary path on stationary workspace. Then, this path is used to guide the RRT growth. The RRT planner validates the ACD's path when a query is established in the actual workspace. If a collision is detected due to the change in the workspace, the planner re-plans the path locally through the changed regions.

Two approaches have been proposed to benefit from this combination. The planners focus on the enhancement of navigation problem for omnidirectional robots in partial dynamic workspace. In next sections these two proposed methods are discussed in more detail.

4.1 RRTs Validator Planner

The RRT validator uses ACD's path as guidance to RRT tree's growth. It considers the path's points as an ordered set, and directs the bias of the tree toward these vertices. The RRT trees branch toward these set in the same order, point by point. In the initial state, the probability of choosing the next point of the path is set to the value of 100 %. If a collision is detected, then the probability is reduced in order to allow the RRT explores the free space and attempts to reconnect to original path's point. If it reconnects, then the probability to choose the next point is set again to the value of 100 % to force the planner follows the original ACD's path once again.

This strategy forces the planner to follow the guiding path when it is possible, and at the same time, it gives the planner a freedom to find an alternative local path to the collided segments.

In this paper, two RRT validators are used to validate the path. The first one rooted at the initial position and the second one rooted at the goal position. They try to follow the ACD path, or find an alternative local path. The RRT trees are shown in Fig. 5-a, where they try to follow the ACD' path (the dotted line).

4.2 Local RRTs Planners

The second proposed planner uses simple straight-line planner to connect the ACD path's points and to test the collision. The planner tracks the valid points of the path and creates sequences of these points. In case that all points are valid, then the planner returns these points as a solution for the planning problem. In the other case when the workspace is changed, and a collision happened, the planner breaks the original path sequence on the collided locations and creates multi-sequences of the continuance valid points. It also excludes the points that are located in obstacle areas.

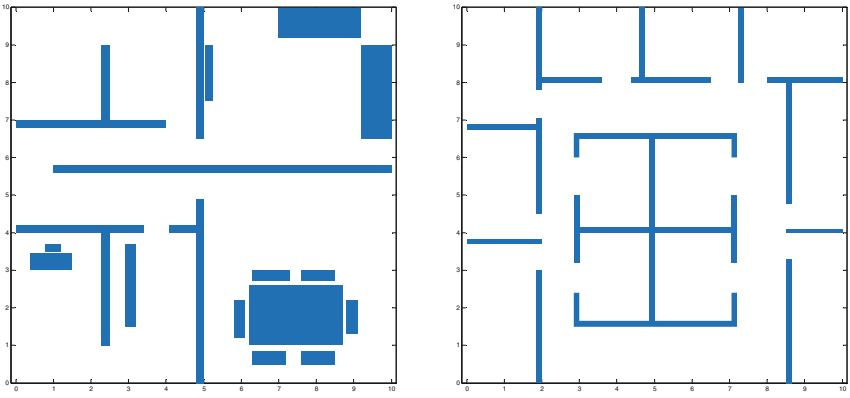
Each of these sequences is associated with RRT tree. The trees explore the space freely with small bias toward the other tree's nodes. When two nodes are connected, the corresponding trees are merged. When all trees are merged, they form a single tree contains the initial and goal locations.

In this planner, our strategy is to generate augmented local RRTs, in order to navigate around the new obstacles locally. Figure 5-b shows the local RRTs method in simulation. In this example, it creates nine local RRT trees based on the ACD path, which generated in the stationary workspace.

5 Tests and Results

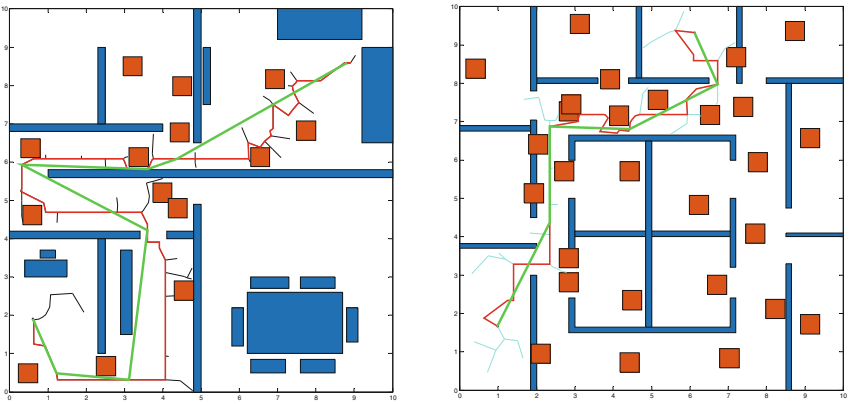
The proposed approaches are tested in two different workspaces as shown in Fig. 6. The first workspace represents an office with one route between the rooms, and the second one represents offices, which have two possible routes between them.

The robot in this work is considered a holonomic point moves in the workspace. The results of the proposed methods are compared to the other methods, i.e. the basic RRT algorithm, Goal Bias RRT, and the bias toward the other trees. Figure 7 shows an example of RRT path generated by the proposed methods in the testing workspaces, where in (a) the local RRTs method is used, and in (b) the RRTs validator method is



(a) (b)

Fig. 6. Testing workspace, a: office has one route between the rooms (WS1); b: two routes between offices (WS2)



(a) (b)

Fig. 7. Example of RRT path using a: Local RRTs method; b: RRTs validator method, on partially changed workspaces. The bold lines represent the RRT path and the shorten one in both tested workspaces. The boxes represent the new obstacles

used. The bold line represent the shorten path to RRT's one. Where the algorithm which generates the shortened path is proposed in [18].

5.1 Testing Parameters

The bias values, which are given to the compared methods, are set as shown in Table 1, where the basic RRT chooses a random point without any bias. The goal-bias RRT directs the growth of the tree toward the goal location by selecting this location in probability of 10 %. In the tree's nodes bias, the RRT chooses a point of the others trees by the probability of 30 %, which force the trees to merge more quickly.

Table 1. The probability of choosing next points (The bias value).

Method	Bias value
RRT	0
Goal bias	0.1
Tree node bias	0.3
RRTs validator (valid point)	1
RRTs validator (Collison)	[0.2,0.1,0.7]
Local RRTs	0.3

In our proposed methods, the bias value of the validator RRTs is set to 100 % when no collision is occurred. Otherwise, it has the value of 20 % of the bias toward the next valid point in the ordered set. In addition, the value of 10 % to bias toward any other points in those points set. The planner in this case has the probability of 70 % to explore the workspace freely and biases the growth toward randomly chosen samples. The last method, local RRTs approach, uses the bias toward the other trees by the value of 30 %.

The simulation is repeated 100 times and the average of successful tries are considered when comparing the results. The results include the execution time; the number of RRT iterations which corresponding to the number of RRT's branching attempts; and the number of successful attempts to find a path.

The probabilistically completeness value is estimated using the number of successful attempts. While the efficiency value is estimated using the time of execution and iterations results. The time of execution could vary significantly, based on the hardware and code optimization, while RRT iteration is independent of HW and the programmers skillful.

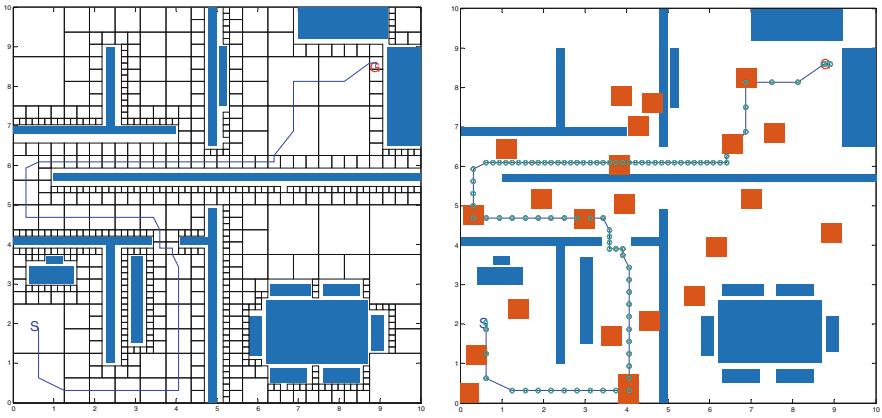
The ACD resolution is set to be 0.2 unit. The ACD's path points are generated in ordered form, from the initial to the goal locations. They are constructed using the initial and the goal points, the free cells' centers, and the barriers' midpoint between the consequence cells.

We use the Dijkstra algorithm to search in the ACD's graph. The RRT parameters are set as follow; the extension step is equal to 0.3 unit. And, the bias value is fixed at the probability of 100 % for next path' points in case of no collision is detected, and it is reduced when the path is collided within obstacles. The reduced value is divided into three parts. 1- The bias toward the next valid point is set to the value of 20 %. 2- The bias toward other path's points is given the value of 10 %. 3- The rest of bias is relaxed to allow the planner chooses random samples freely. The RRT result is considered as failed, if it cannot find a path after 2000 tries of branching.

5.2 Results and Discussions

In the first workspace, new obstacles are scattered on the original workspace. They are positioned to collide the ACD's path and add more difficulty to navigate through the changed workspace. The workspace's changes are shown in Fig. 8-b, where the boxes represent the new obstacles. The ACD's path is shown as a solid line between the initial and the goal locations. The cycle markers represent the bias points. ACD algorithm approximates the free cells as shown graphically in Fig. 8-a, the path in this case is produced using the Dijkstra searching method in the ACD's adjacency graph.

The numerical results are shown in Table 2, where the proposed methods show a probabilistically completeness. The local RRT method gives the best results in term of efficiency; it has the lowest execution time, and the lowest iteration to find a path. Moreover, the RRT validator method gives a better result comparing to the other competitors. Figure 10-a sums up the iteration results for the first workspace WS1 using the boxplot representation.



(a) (b)

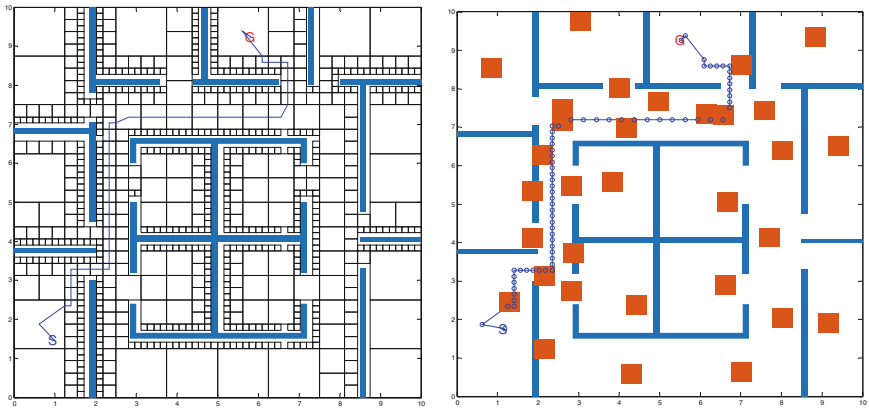
Fig. 8. Offices-like workspace (WS1), a: The approximation decomposition of the free region; b: the new obstacles are represented as boxes. ACD' path is represented by the solid line, and the cycle markers represents the bias points

Table 2. The result of the tested methods on WS1.

Method	Mean time [Sec]	Mean iteration	Success [%]
RRT	1.03	1137.11	96
Goal bias	1.12	1180.57	87
Tree node bias	1.23	1365.34	80
RRTs validator	0.45	270.19	**100**
Local RRTs	**0.19**	**95.20**	**100**

In the second workspace, the changes are introduced by scattering new obstacles in the stationary workspace. The new obstacles are collided within the ACD's path, and they produce more narrow passages. Figure 9-b shows the changes in the workspace, where the new obstacles are represented by boxes. The ACD's path is shown in the figure as solid line between the initial and the goal locations. The bias points, which are generated based on this path, are shown in the figure as cycle markers. Figure 9-a, shows the approximation of the free workspace using ACD algorithm.

The numerical results are shown in Table 3, where the proposed methods give the best results; they are probabilistically complete as we infer from the success rate result. Moreover, the local RRT method gives the best results in term of efficiency; it has the lowest execution time, and the lowest iteration average. Figure 10-b condenses the iteration results for WS2 using the boxplot representation.

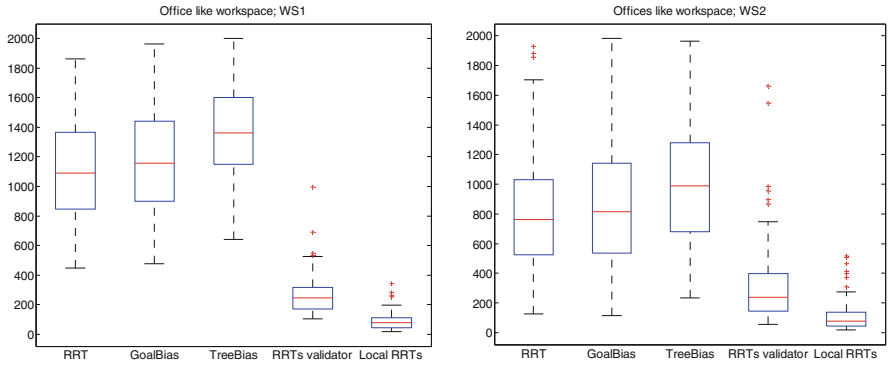


(a) (b)

Fig. 9. Offices-like workspace (WS2), a: a graphical representation of approximation cell decomposition; b: the new obstacles (boxes). ACD path represented by the solid line, and the bias points represented by cycle markers

Table 3. The result of the tested methods on WS2.

Method	Mean time [Sec.]	Mean iteration	Success [%]
RRT	0.92	817.13	96
Goal bias	0.98	871.06	94
Tree node bias	1.076	1005.10	86
RRTs validator	0.62	332.07	**100**
Local RRTs	**0.24**	**117.17**	**100**



(a) (b)

Fig. 10. RRT's iteration boxplot for WS1 (a) and WS2 (b)

6 Conclusion

In this work, the approximation cell-decomposition algorithm is combined with the RRT planner in order to enhance the omnidirectional robot's navigation on partially changed workspace. The ACD finds a spatial path on preliminary and stationary workspaces, and then the RRT is used to validate this path on the actual workspace.

Two methods have been proposed in this paper. First, the planner creates instances of RRT, which bias toward the ACD path's points in order form. It updates its bias value based on the collision detection information.

The Second method uses a straight-line planner to connect path's points and creates local RRT trees on both sides of collided segment of the path. The proposed methods compared with the other approaches. The simulation results shows that the suggested methods give the best results in terms of completeness, in addition, the local RRTs method gives the best result in terms of efficiency in the both workspaces.

References

1. Choset, H., Lynch, K.M., Hutchinson, S.: Principles of Robot Motion: Theory, Algorithms, and Implementation. MIT Press, Cambridge (2005)
2. De Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry. Springer, Heidelberg (2008)
3. LaValle, S.M.: Planning Algorithms. Cambridge University Press, Cambridge (2006)
4. LaValle, S.M.: Rapidly-Exploring Random Trees: A New Tool for Path Planning (1998)
5. LaValle, S.M., Kuffner, J.J.: Rapidly-exploring random trees: progress and prospects. In: 4th Workshop on Algorithmic and Computational Robotics: New Directions, pp. 293–308 (2000)
6. Kuffner, J.J.J., LaValle, S.M.: RRT-connect: an efficient approach to single-query path planning. In: Proceedings of 2000 ICRA Millennium Conference IEEE International Conference Robotic Automation Symposium Proceeding 2, vol. 2, pp. 995–1001 (2000)
7. Strandberg, M.: Augmenting RRT-planners with local trees. In: 2004 IEEE International Conference on Robotics and Automation, Proceedings of ICRA 2004, vol. 4, pp. 3258–3262 (2004)
8. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 2383–2388 (2002)
9. Bruce, J., Bowling, M., Browning, B., Veloso, M.: Multi-robot team response to a multi-robot opponent team (2003)
10. Abbadi, A., Matousek, R.: RRTs review and statistical analysis. Int. J. Math. Comput. Simul. **6**, 1–8 (2012)
11. Sleumer, N.H., Tschichold-Gürman, N.: Exact Cell Decomposition of Arrangements used for Path Planning in Robotics (1999)
12. Abbadi, A., Matousek, R., Osmera, P., Knispel, L.: Spatial guidance to RRT planner using cell-decomposition algorithm. In: 20th International Conference on Soft Computing MENDEL (2014)
13. Van den Berg, J.P., Overmars, M.H.: Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. Int. J. Robot. Res. **24**, 1055–1071 (2005)

14. Katevas, N.I., Tzafestas, S.G., Pnevmatikatos, C.G.: The approximate cell decomposition with local node refinement global path planning method : path nodes refinement and curve parametric interpolation. J. Intell. Robot. Syst. **22**, 289–314 (1998)
15. Lingelbach, F.: Path planning using probabilistic cell decomposition. In: Proceedings of IEEE International Conference on Robotics and Automation ICRA 2004, vol. 1, pp. 467–472 (2004)
16. Cai, C., Ferrari, S.: Information-driven sensor path planning by approximate cell decomposition. IEEE Trans. Syst. Man Cybern. Part B Cybern. **39**, 672–689 (2009)
17. Abbadi, A., Prenosil, V.: Safe path planning using cell decomposition approximation. In: International Conference Distance Learning, Simulation and Communication, Brno (2015)
18. Abbadi, A., Matousek, R., Jancik, S., Roupec, J.: Rapidly-exploring random trees: 3D planning. In: 18th International Conference on Soft Computing MENDEL 2012, pp. 594–599. Brno University of Technology, Brno (2012)